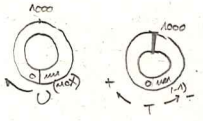
SPCA-BULLET PTS

Simone Guggiari

1 INTRODUCTION
- int – floats — rng/ordering properties
- know assembly
- memory matters – RAM, performance levels
- performance — constants
- don't just execute — I/O/network

2 INTRO TO C
- macro pre processor — string/file substitution, conditional compilation — no OOP features
- DMA (direct memory access) – using pointers / not automatic memory management
- workflow
- GNU gcc toolchain
- control flow — if/else/switch/return/for/while/do/break/continue/goto
- basic types — default signed
  - int bool void — type, no value – with functions = procedures
- operators
- arrays
  - strings
- word size (dimensione naturale dati usati da un computer)

3 REPRESENTING C INTEGERS
- bit wise operators
  - & | ^ << ~ — shift: undefined behaviour
- int ranges
  - U/T min/max — constant/default = signed; mixing: sign → U
- casting
- addition + multiplication * (U/T) — some bit-lvl behaviour
- shifts for mathematical op — rounding in wrong direction



4 POINTERS
- stack + heap
- mem address (ptr)
- segmentation fault
- address space randomized
- pointer arithmetic
- arrays & pointers — A[i] = *(A+i)
- passing reference / dereferencing — //pass value = copy
  - copies & examples
  - *strcpy

```
int v = 5;
int *ptr = &v;
*ptr = 6;
```

(dereferencing NULL ptr) — NULL = void * (0x0000)

5 DYNAMIC MEMORY ALLOCATION
- C memory api (alloc + dealloc + realloc)
  - malloc()calloc()free() realloc() — good practice
  - size_t (= ptr_diff_t)
- memory corruption
- managing heap
  - bookkeping data
  - memory leaks (footprint ↗)
- structures
- unions
- type definitions

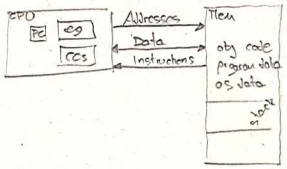
- statically = GLOBAL (program)
- automatically = LOCAL (function)
- dynamically = EXPLICIT

→ declared in stdlib.h

- variables and types
- dynamic/generic DS
  - linked lists, ...

6 C WRAP UP
- C preprocessor
  - header files (inline) = API (primitive) — ".h" / <.h>
  - macro definitions
  - token based — swallow semicolon
- modularity
  - compilation units (static/extern) — declaration/definition
  - interfaces
- function pointers (delegates)
- assertions
- goto (recovery code)
- setjmp() longjmp() – env → used for error handling
- coroutines — Coro_new, –free, –start, –switchTo (+ initialization)
  - parser — +closures — decompressor +

7 BASIC X86 ARCHITECTURE
- ISA (instr. set)
  - architecture/microarchitecture
- CISC/RISC
  - advantages/differences
  - orientation — stack/regs
- machine code — FC
  - reg/counter/condition codes
  - address/data/instruction
- memory
  - byte/word addressable
- code operations — : arithm op, : data transfer (sw/lw), : transfer control (branch/jmp)
- assembler / disassembler + linker
- ARCHITECTURE — mov
  - immediate register memory
  - addressing modes (normal/displacement/index)
  - lea — D(Rb,Ri,S) → Mem[Reg[Rb] + S·Reg[Ri] + D]; S → 1, D → 0 (assumed)
  - operations/arithmetic
  - comparison/conditional codes (flags)



*see RefSheet

8 COMPILING C CONTROL FLOW
- if then else — efficient method without repetition
  - goto, conditional move, load effective address (lea)
- do while loops
- while loops → efficient method without repetition
- for loops
  - translation to while
- switch
  - compact (jump table), jumps
  - sparse (binary tree)

mono

- procedure call & return
  - stack (bottom, top, addresses, growing)
  - push, pop, stack ptr
  - stack frame (context)
  - arg build, loc var, saved reg, old frame ptr
  - return address, call arg
- calling conventions
  - caller, callee, save
- red zone
- !long swap – non leaf
- volatile -> tell compiler not to optimize this

base ptr : space of reg
stack ptr : where it points

9 COMPILING DATA STRUCTURES

- one dimensional array
  - contiguous region
  - arithmetic to access
- nested arrays (different / fixed size)
  - row/col major A[i,j] = Mem[A + i*(C*K) + j*K]
- multi lvl arrays A[i,j] = Mem[Mem[A + i*sizeof(ptr)] + j*K]
  - strengths, limitations 2 mem reads / dynamic resizable / easily (multiplication) (syntax is the same)
- structures
- alignment
  - gaps, K, largest element, order (save space - big first)
- arrays of structures (-> primitive data size)
- unions (largest elem)

10 CODE VULNERABILITIES

- worms, viruses (run independently / propagate)
- stack overflow bugs
  - limit char size/signed casting
  - overwrite ret address
  - exploit
- stopping bugs
  - randomize stack
  - read-only code (non executable)

Δ scanf, sleeper, gets -> fgets, strcpy

11 MEMORY ALLOCATION

- the problem
  - implementation issues
- allocator (immediately + aligned)
- throughput/latency
- peak memory utilization
  - aggregate payload (currently used = Σ malloc() - free())
  - current heap size (stack)
- fragmentation
  - internal/external
- Implicit list using length (1) (aligned -> use low order bits for allocated flag) (fewer bytes left over)
  - first/next/best fit
  - splitting
  - coalescing + boundary tags (immediate / deferred) (bidirectional)
- Explicit list among free blocks (2)

manual allocation
- explicit
- implicit (GC)

$U_k = \max_{j \le k} P_j / H_k \in [0,1]$ (mac handlers exp)

- use ptr (still use boundary tags)
- linear search
- policy
- lifo/address ordered < pro: simple, con: worse fragm | no search
- Segregated free list (3) (size class) / groups of pow(2) (logarithmic time)
- Blocks sorted by size (4)
- GC (garbage coll)
  - strong typing -> no ptr reference
  - root nodes (ref in heap -> heap) like reg/stack
  - mark and sweep alg (code)
- memory pitfalls
  - dereferencing bad pointers (scanf)
  - reading uninitialized memory (malloc)
  - overwriting memory
  - referencing nonexistent variables
  - freeing blocks multiple times
  - referencing freed blocks
  - failing to free blocks
  - memory leaks (free only perf)
- gdb (debugger)

wrong alloc size
OBO
no str size check (gets()) -> stack overflow bugs
bad ptr arithm
ref ptr instead *ptr
es bool on call

-> good practice: := NULL

12 LINKING

- compiler driver
  - static linking
  - object files
- modularity (small files)
- efficiency (re-compile only part)
- linkers:
  - symbol resolution -> symbol table (name/type/size/loc) (private (int) globals @ 0x...)
  - relocation

Δ order in finding def matters!

ref -> def

- object files
  - relocatable .o
  - executable .exe
  - shared .dll / .so
- ELF file format
  - head/segment (setting info) K, byte ordering, file type, machine type/... // page size, VM address space,
  - text/rodata/data/bss/symtab/reltext/reldata/debug
- linker symbols (-> names & loc)
  - global
  - external
  - local (static)
  - weak/strong (+ rules)
- static libraries .lib
- shared libraries .dll
- load/run- time linking

Δ order in which linker looks (for unresolved external references)

-> dynamic)

13 FLOATING POINT

- representation
  - s-exp-frac

- interpretation
  - S-E-M
- fractional numbers x/2^k
- IEEE 754
- numerical form/encoding
- precision
  - single/double/extended/quadruple
  - exponent ranges
- casting/converting
- normalized
- denormalized
- special values
- ranges
- rounding
  - zero/down/up/even
  - normalize/postnormalize
- addition, multiplication
- puzzles

14 OPTIMIZING COMPILERS

- strong suits
  - mapping/reg alloc/dead code elimin
- pitfalls
  - asymptotic/mem aliasing/side effects
- limitations
  - conservative constraints
- how to optimize
  - alg/data represent/procedures/loops
- removing unnecessary procedure calls and checks
- code motion and precomputation
- strength reduction
- common subexpressions
- optimization blocker
  - procedure calls
  - memory aliasing
- blocking and unrolling

15 ARCHITECTURE AND OPTIMIZATION

- recap on sequential/pipelined processor design
- superscalar processor design
- ILP (instr lvl parallelism)
- superscalar performance
- micro ops
- dataflow view
- branch prediction
  - ICU instr control unit
  - EU execution unit
- space-time tradeoff

- unrolling/reassociation/multiple accumulators
  - original
  - loop unrolling
  - -with reassociation
  - -with separate accumulators

16 CACHES

- hit/miss
  - rate/time/miss penalty
- policies: placement/replacement
- cache misses
  - cold(compulsory)
  - conflict
  - capacity
  - coherency
- memory hierarchy
  - reg/L1/L2/L3/mem/SSD
  - SRAM/DRAM/HD(disk)/remote
- locality
  - spatial/temporal
  - advantages
- cache organization
  - C-B-E-S
  - address word
  - tag, set index, block offset
  - associativity
  - valid bit
- cache reads
  - direct mapped / E-way associative / fully associative
- cache writes
  - write-hit: write through/back
  - write-miss: (no) write-allocate
- cache optimization
- blocking

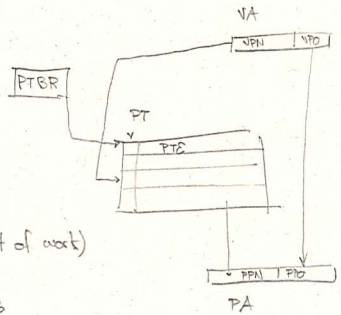
17 EXCEPTIONS

- event
- control flow
- system state change
- exception vectors
- kernel mode
- synchronous exception
  - traps/faults/aborts
- asynchronous exception
  - interrupts/IO/maskable
  - hard/soft resets
  - how they work
- interrupt controllers
- programmable/pin/vectors

- simultaneous/prioritize/mask

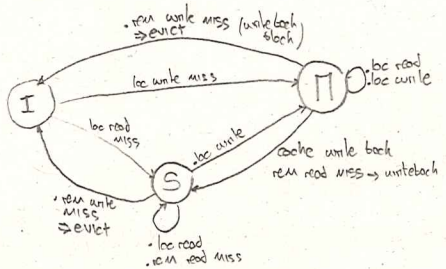
18 VIRTUAL MEMORY

- problems (too much mem / what goes where)
  - protect/share (disjunct/conjunct)
- solution: address translation
- linear/virtual/physical address space
- MMU (mem management unit)
- PT/PTE/PTBR (page table/entry/base reg)
- VPN/VPO – PPN/PPO
- page fault / paging → divide mem in pages (min unit of work)
- uses of VM
  - caching (+working set) → active virtual pages
  - memory management
  - simplify linking/loading (all addresses similar)
  - memory protection (+SIGSEGV) ↳ permission bits
- address translation (see steps)
  - page hit
  - page fault
- TLB (translation lookaside buffers)
  - simple mem example (+case study) ⇒ see
- multi-lvl page tables (hierarchical PT)
  - page/frame (content/container)
  - formula AS/PS*PTE=PT
  - TLB-T/I, C-T/I/O
  - page – table (map/directory ptr/directory)
  - master page table
- flags in PTE
  - AVAIL/G/PAT/PCD/PWT/US/D/A/RW/P
- large/huge pages
- optimizing TLB



19 MULTIPROCESSING

- consistency (=order of op) SMP
- coherency (all values in cache same)
- program order
- visibility order
- sequential consistency
  - pro/con (programmer correct analyz | fast cant reorder + combine; Serialization is response)
- cache coherence with snooping
- dirty data/bit
- MSI (modified, shared, invalid)
  - state machine/protocol
  - issues
- MESI (exclusive)
  - hit/rdx
- MOESI (owner) - AMD
- MESIF (forward) – intel



- relaxing sequential consistency
  - w2r, w2w, break w atomicity, weak ordering
- Barriers and fences
  - compiler/memory → prevents CPU
- multicore synchronization
  - TAS (test and set)/DTAS
  - CAS (compare and swap) / CAS (compare and set) ↳ return boolean
  - ABA problem
- simultaneous multithreading (SM)
  - bottleneck/data dependencies
- architectures types
  - SMP (symmetric multiprocessor system) (this)
  - NUMA (+cache coherence)
- performance implications
  - # mem latency / cache address latency / false sharing

```
void acquire(int *lock){
    while(TAS(lock) == 1);
}
```

```
void release(int *lock){
    *lock = 0;
}
```

```
int CAS(int *p, int old, int new){
    int value = *p;
    if(value == old) *p = new;
    return value;
}
```

load check/set return

- LL multithreading synchronization
- SC /lock free atomic read-modify-write op

20 DEVICES

- bus/regs/interrupts
  - reg access (mem mapped/IO/indirection)
- UART (universal asynchronous receiver/transmitter)
- PIO (programmed IO) - polling
- dealing with caches (bypass it)
- DMA (direct memory access) × PIO, bypass CPU
  - transfers directly
  - fast network/dirk interfaces
  - controller
  - advantages
  - cache pollution (+solutions), inconsistency
  - IOMMU (for VM)
- device drivers
- Buffer rings/descriptor rings (ptr)
  - packets
- producer/consumer queue
  - overruns/underruns
- transmit
  - device has no packets
  - CPU has no slots
- receive
  - device has no buffers
  - CPU read all packets
- device initialization
- IO state machine (Running/Stopped/Waiting)
  - hw/sw side
- PCI (peripheral component interconnect)
  - discoverable busses
  - device discovery/address allocation
  - interrupt routing/intelligent DMA (bus mastering)
  - interrupts (MSI)
- DMA reads/writes (?)
  - before/after

- obtain/set status
- get input/set output reset status

Dir DevReg Rd Dev IRQ Shared Mem